

ScripTale: Generation of Procedural Narrative

Bachelor's thesis

Design and Development of Video Games

Author: Ripoll Galan, Laura

Director: Torelló Oliver, Josep

CITM (UPC) - Terrassa

Course: 2017 - 2018

Scheme: 2014

Abstract

The subject of research is the procedural narrative. Procedural narrative is the way of generating narrative automatically along with the progress of the game. More specifically, the subject of the study will be about exploring the way procedural narrative could be implemented in a project and through with a tool external to the code of the game itself.

Key words and notation

AI: Artificial Intelligence

PN: Procedural Narrative

PCG: Procedural Content Generation

LL: Linear Logic

GOAP: Goal Oriented Action Planning

States: Resultant variables of each character after an action

Conditions: Rules to be ascertained to develop an action

Actions: Performance of an available option of the character

Messages: Content of the dialogue said by a character towards another

Index

Abstract	1
Key words and notation	1
Index	2
Index of figures and images	5
Index of tables	6
1. Introduction	7
1.1. Project justification	7
1.1.1. Motivation	7
1.1.2. The problem	7
1.2. Aim of the project	8
1.2.1. General goals of the project	8
1.2.2. Specific goals of the project	8
1.3. Project scope	9
2. State of the art	10
2.1. NPCs. Learning the player's way of play	11
2.2. Voice recognition and realistic graphics	12
2.3. Procedural Content Generation	14
2.4. ANGELINA	15
2.5. Procedural Narrative	15
2.6. The tool: ScripTale	17
3. Planning	19
3.1. Gantt chart	19
3.1.1. Tasks description	19

3.1.2. Chart	22
3.2. SWOT	24
3.2.1. Analysis	24
3.2.2. Contingency plan	26
3.3. Costs analysis	27
3.3.1 Costs overview	27
3.3.2. Costs breakdown	30
3.4. Management tools	33
4. Methodology	33
4.1. Tracking of the project	34
4.2. Validation tools	34
5. Development	36
5.1. Design of the tool	36
5.2. ScripTale Initial implementation	40
5.2.1. Curve Editor	40
5.2.2. States Editor	42
5.2.3. Relationships Editor	45
5.2.4. Dialogue System	46
5.2.5. Characters	47
5.2.6. Messages and actions	48
5.2.7. Emotions	49
5.3. Initial workflow of ScripTale	52
5.4. Last state of ScripTale	53
5.4.1. Curve editor	53
5.4.2. States Editor	54
5.4.3. Relationships Editor	55
5.4.4. Dialogue System	56
5.4.4. Characters	56

5.4.5. Messages and Actions	60
5.4.6. Emotions	60
5.5. Final workflow of ScripTale	61
6. Conclusions	62
Bibliography	65
Annexes	68

Index of figures and images

Monte Carlo Tree Search Algorithm schema	Fig. 2.1.
Characters' decision taking schema in <i>The hare and the turtle</i>	Fig. 2.5.
Gantt diagram	Fig. 3.1.2.
Design schema of the base classes of <i>ScripTale</i>	Fig. 5.1.1.
Design shema of the editors of <i>ScripTale</i>	Fig. 5.1.2.
Curve editor of <i>ScripTale</i>	Fig. 5.2.1.a.
Curve functions of <i>ScripTale</i> necessary for the programmers	Fig. 5.2.1.b.
Schema of the States functionality	Fig. 5.2.2.
Schema of the Relationships functionality	Fig. 5.2.3.
Workflow of <i>ScripTale</i>	Fig. 5.3.
Relationships Editor of <i>ScripTale</i>	Fig. 5.4.3.
Current workflow of <i>ScripTale</i>	Fig. 5.5.
Simple example of use of <i>TeLLer</i> provided by its creators	Fig. 6.

Index of tables

Work Packages	Table 3.3.1. a
Partners	Table 3.3.1. b
Budget	Table 3.3.1. c
Human resources	Table 3.3.2 a
Software	Table 3.3.2 b
Hardware	Table 3.3.2 c
Indirect costs	Table 3.3.2 d

1. Introduction

1.1. Project justification

1.1.1. Motivation

The strongest justification of this project is my intrinsic motivation to link two of my favourite fields in video games: Artificial Intelligence and narrative. Not only because of my own interest in these two areas but also due to the high rate of difficulty involved and the amazing result that could be achieved by defeating the challenge to unify both opposed behaviours, as narrative tends to be structured and AI emergent.

1.1.2. The problem

Nowadays, procedural methodologies in video games are spreading, from animations to terrain creation and so, narrative. However, there is not any global tool in Unity's Asset Store to allow procedural narrative generation in games developed with this popular engine.

More investigation in this field is needed and a high demand of procedural narrative exists in the industry. Some AAA games try to make such an approach but, these implementations are not public and other developers could find it difficult to include such a valuable feature on commercial engines like Unity.

Some theoretical research has been carried out, however, there is no practical tool to apply procedural narrative to real projects.

1.2. Aim of the project

The goal of the project is to program a tool in Unity that could be implemented in any game to generate procedural narrative through a dialogue system with the existing characters.

The tool will be submitted in a Unity package format to make it easier to put it in the Asset Store afterwards. If the tool has proved an interest in the community, a Demo will be delivered as a Unity project so the code will be accessible to anyone and also in a compiled game version. However, it is not an aim of this project but a further possible work beginning with the tool.

To develop the entire project, be it the package of the tool, the engine to be used will be Unity. On the other hand, for scripts edition and code, Visual Studio will be the IDE to work with.

1.2.1. General goals of the project

- **Generate a narrative with different results at each game:** Make it possible to generate different narrative with a same base system dialogue but with unexpected results at each game.
- **Approach the tool to independent developers:** Make a available tool to improve narrative systems on independent games in the market.
- **Get the tool published in Asset Store:** Accomplish all the requirements to publish the result of this research, the tool, to the Asset Store in unity.

1.2.2. Specific goals of the project

- **Control an AI with unpredictable results:** Being capable of controlling most of the internal paths or processes of the AI without losing the surprise of getting an emergent narrative situation.

- **Get an editable tool:** Give as much freedom as possible to the designer or the screenwriters. Have a user friendly tool to allow most of the editing by Unity Editor and avoid programmers spending time on it through code.
- **Reach the specific target pointed:** Reassure that the target established is interested on a tool of this type by checking the downloads and further projects developed under the tool's license.
- **Get an entry to the market:** Achieve a recognisable brand as much as possible. Get a strong tool to validate an advance on the AI applied in narrative and spread this innovative feature within the market.

1.3. Project scope

The target whom the tool is addressed to, is a very specific segment in the videogames industry: independent developers. Independent developers do not usually own the technical resources neither have the time to apply to their game such a feature that could be more easily found in AAA games. That is why a huge amount of the so called *indies* don't even use their own engine, they take advantage of third parties. Offering a specific tool with a modest price or, in this particular case for free, looks appealing to those factions looking for cheap tools to improve the efficiency or the quality of the project.

Due to this specific target and the objectives established for the current project, the nature of the tool that has been developed is controlled. It needs to have a controlled environment despite creating emergent "random" stories. Nevertheless, it will give control to some extent to the designer.

A future project could try to remove the designers' input and create a tool to generate an entire story with much less external input. Possibilities starting from the current study are endless. Removing the human interaction and improving the AI would be just the first step. From this entry point technology could improve, leading to more sophisticated results like programming a writer tool, a machine that could be able to write and make up stories by itself.

For now, the first approach beginning with this project's final result could be making the tool intelligent enough to take control of the intensity curve. Another further implementation that would run along parallel lines would be to get the tool itself to know if the dialogues match, without requiring external restriction to get it to work properly.

2. State of the art

Nowadays, it seems AI in video games has been occluded by the AI improvements in other areas such as the way people interacts with machines or devices. However, video games is the category in which AI can show its full potential easily. That is why, having the players been used to already seen great techniques applied to games, the gaming industry remains relatively aside, working on AI improvements which are yet taken for granted by a community that gets more amazed by advances in other fields.

Video games have been working on machine learning, context-sensitive behaviour, neural networks and many other implementations since 2000 and are still improving the AI on the sector. One of the best references of games applying machine learning in that time was *Black and White*, launched in 2001.

In that first referent for the AI development in video games, the player was given a creature to bring up. It behaved like a kid, curious yet inexperienced and afraid of the outside world.

Machine learning systems have not stopped evolving and it is very likely to perceive great changes in the video games over the upcoming years.

2.1. NPCs. Learning the player's way of play

The most common role when talking about AI in video games is the control of the non-player characters (NPCs). It is also one of the most overlooked by the players, who only see the AI implementation in form of NPCs if they don't look intelligent, which would be their aim. With the intention of making the NPCs behave in a credible way, designers normally use the so-called Finite State Machine algorithm. It had already been introduced in video games back in 1990. It takes into account all the possible situations the NPC can encounter and programs a specific reaction to them. It would react to the player's action with a pre-programmed behaviour.

Another advanced technique applied in the NPC strategy is the Monte Carlo Search Tree (MCST). This algorithm uses random trials to solve a problem. It could, for example, be used for movement. The NPC would consider all its possible moves, then all the possible player moves in response, all its possible responding moves and so on. This looping process could expand indefinitely in several branches, that's why the method is based on a Search Tree. After a settled number of times repeating the process, the NPC would calculate the payback and then the most suitable branch to follow. After the players' reaction, the AI would start the search tree again.

However, in this early stage of video games in 1990s, NPCs didn't get to look intelligent at all because they could fail over and over again in specific situations. At that point, AI research in video games started to focus in the ability to learn, which was an important feature that was lacking in NPCs. AI would evolve and learn from the input of the player. One of the first and best examples to give learning capabilities to a NPC is the game *Petz* (Ubisoft, 1995). The player was able to train a digital pet such as a dog or a cat and the pets' behaviour changes along the bond developed with the player. It learns from the user and that leads to a more unique and personalized gameplay.

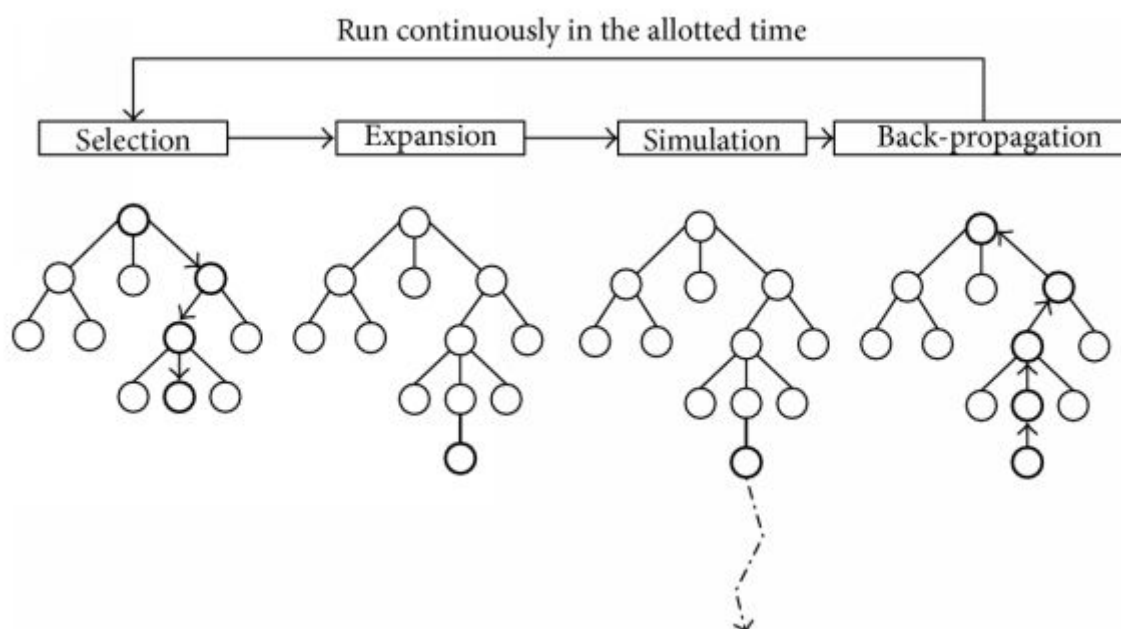


Fig. 2.1. Monte Carlo Tree Search Algorithm schema.

Recent Advances in General Game Playing - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/Four-steps-of-the-Monte-Carlo-Tree-Search-algorithm_fig11_282043296 [consult 26 February, 2018]

2.2. Voice recognition and realistic graphics

The common associations society does with AI is the present in the daily life feature, voice recognition. Voice recognition is mainly applied on assistants, the most well-known ones being: *Siri*, *Cortana*, *Google Assistant* and *Alexa*.

These assistants are also changing the way players interact with video games, making it easier and more intuitive. *Destiny 2* (Activision, 2017), for instance, uses *Alexa* for its hand-free voice commands feature. These commands are thought to enhance the user experience and make it more comfortable. It offers more than a thousand costume voice lines and allows, for example, the player to equip Crucible, Nightfall, Strike or Trials of the Nine.

Another game that can be supported by a voice recognition assistant is *The Inspection Chamber* (BBC R&D, 2017). Available to play with *Alexa* and soon with *Google home*, the users play their part through voice interaction. In this case, it is appreciated that the game is entirely based on the voice recognition that AI has brought in.

In spite of the good reputation AI has globally achieved with the voice recognition systems, the most impressive one in which it is evolving, is the graphics.

One of the most noticeable improvements in modern AI for video games is spotted in the visual aspect. As the Unreal Engine developer, Tim Sweeney (2017) explained in an [interview](#) that advancements in AI can improve the characters' expressions and movements. "I think there's a lot that can be gained from incorporating more advanced and modern forms of AI into our game processes. Especially in the areas where we can use a huge amount of data to have algorithms figure out how humans act in the real world and extrapolate that in the game world," (Sweeney, 2017).

It is possible to endow personality to the characters by reproducing human behaviour. Video games could transmit more human emotion beyond technologies such as motion capture, scanning hours and hours of real footage and mapping it onto the game. *Hellblade: Senua's Sacrifice* (Ninja Theory, 2017) is one of this examples of motion graphics with an impressive result of realism but, AI could achieve better behaviours and save this huge amount of time spent on it through deep learning. Tim Sweeney believes AI could make gaming improve in terms of visual quality.

The improvements of graphics in which AI can go in conjunction with the virtual reality. Video games are currently moving in direction towards Virtual Reality and Augmented reality, and so is Artificial Intelligence. As AI progresses to get realistic VR technology, immersion in virtual worlds is getting stronger.

Electronic Arts' CEO already [forecasted](#): "Your life will be a video game." (Wilson Andrew, 2017).

2.3. Procedural Content Generation

Procedural generation is a method of creating content as the game evolves. The generation of this content depends on an algorithm and it has no element of randomness. All the result is derived from functions, algorithm and input. The result should then, always be the same. Nonetheless, if the inputs depend on the player's actions, the behaviour of the characters itself or other dynamic elements, PCG could appear to be random and it becomes complicated to keep under controlled the emergent situations to which the procedural generation has led.

Procedural Content Generation is applied to a large range of areas, from the art asset generation to game agents strategy or movement to controlling math. Although it seems unbelievable, math and geometry can also be [procedural](#), as seen in the video Math for Game Programmers: Mixing Geodetic, Hand-crafted and Procedural Geometry (Inc.'s Graham Rhodes, 2015). These PCG techniques have been implemented in games in as many aspects as one can think of. In the [GDC 2017](#), the six developers Tyler Coleman, Zach Aikman, Tanya Short, Tarn Adams, Mitu Khandaker-Kokoris, Luiz Kruehl shared examples of various practical ways of including procedurality in games.

In previous GDC Talks, PCG had already been mentioned; back in 2014, Manuel Kerssemakers, from Abbey Games, compared not only the types of areas that procedurality can improve, but also two algorithm implementations: AI algorithms only serving the game *Reus* (Abbey Games, 2013), and the PCG Algorithm being an integral part of the design in *Renowned Explorers* (Abbey Games, 2015).

PCG is a topic of research which has proved its viability in diverse strands. AI had shown its application in video games beyond Neural networks in *Black & White* (Lionhead Studios, 2001), behaviour trees in *Halo* (Bungie Studios, 2001), and the PGC implementation on map generation brought about *Minecraft* (Mojang AB, 2009).

2.4. ANGELINA

It can be relatively easily conceived that some areas such as art or sound can be modulated and created alongside procedural generation but, Design seems something untouchable that can only be done by human. Thinking up an entire game from scratch is a task that is usually carried out by a designer. Withal, ANGELINA is the practical validation of the potential of PCG, being able to substitute this last task.

Michael Cook is the 30-year-old senior research fellow at University of Falmouth behind ANGELINA. The tool's name stands for "A Novel Game-Evolving Labrat I've Named Angelina." He developed ANGELINA in 2011 and since then, Michael has kept updating the tool. Meanwhile, ANGELINA has developed by itself hundreds of experimental video games. The senior researcher is now working on [Creative Code Generation for Interactive Media](#).

PCG is normally employed in video game development as a tool to aid in a specific feature within a large creative vision that comes from a person. What Michael tries to achieve with ANGELINA is to have a system that procedurally generates the majority or even all the aspects of a game's design by itself.

2.5. Procedural Narrative

Procedural Narrative is the automatic process that generates narrative over time. However, narrative is a structured story by definition so, balancing the emergency of PCG and the structure of the narrative is the main conflict in procedural narrative. Other procedural contents, have no need of a rule or structure to exist but the ones imposed by the designer, whereas narrative does. A story needs to have a beginning and an ending, with its relevant stages and conflict points.

To build an emergent story, characters and AI will take more control of the actions and the main weight of the narrative will be the interaction between characters and

their decision taking aspect. In this way, by letting the player some freedom, the machine and the player will take turns to redrive the story to some desired states established by the Designer (or not).

To illustrate what has been brought up in *Practices for Procedural Narrative Generation* (Martens Chris, 2017), coming up with an schema could make it more clear.

So, a good example of story with a main plot determined by the characters' decisions is the classical tale: *The Hare and the turtle*. As can be seen in the following schema, all the decisions brings the story to an end. Characters can plot their own story:

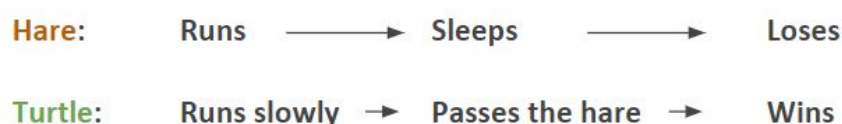


Fig. 2.5. Characters' decision taking schema in *The hare and the turtle*

For the implementation of AI there is not a correct set algorithm to follow, each solution finds its own way: Deep Learning, Machine Learning, Finite State Machines, Behaviour Trees, States, MCSTA... Theoretical studies from various universities, such as the *Development of an Emergent Narrative Generation Architecture for Videogames* (Schudlo, Nicholas A., 2014) point out one shared method for procedural narrative: Linear Logic, a type of substructural logic.

Substructural logic is the study of correct methods of reasoning characterized by following three rules:

- **The contraction rule:** It states that any premise used for a valid deduction, can remain valid only once.
- **The weakening rule:** While a hypothesis of a valid deduction (not a premise) remains valid, more premises can be added to it.
- **The exchange rule:** Also called the permutation rule, it defines that the order of premises is irrelevant to the validation of a deduction.

Linear Logic is one exception in substructural logic due to its omission of the contraction rule and the weakening rule.

Linear Logic emphasizes on the role of formulas as resources. To avoid focusing on the truth or the proof from classic logic, it does not allow the application of the contraction rule or the weakening rule to all the formulas but only to those marked with certain modals. This way it has an involutive negation while keeping strong the interpretation. Provided this focus on resources, Linear Logic is applied in programming such as, in this case in procedural narrative which, as already seen is based on rules and algorithms to control it.

2.6. The tool: *ScripTale*

ScripTale, name of the tool which is object of the current study, will begin from the point of the theoretical investigations done in Procedural Narrative. That means it will use the information asserted in these studies to progress.

It will go further implementing in Unity what these studies have shown in papers. Despite some AAA games such as *The Witcher 3: Wild Hunt* (CD Projekt RED, 2015) could have presented similar features or innovative narrative components, none of them makes the code available and/or scalable, being closed and specific for the game. The distinctive feature of *ScripTale* is the freedom, not only for offering the

tool to a wider range of studios but also for being able to adapt to any game made with Unity.

This AI tool's limitations yet would be positively reviewed for its specific purpose. *ScriptTale* depends on human input to work, the designer or the screenwriter. The responsible of the narrative task would have to provide the tool with a minimum amount of dialogue classified on the type of character that would say that sentence and in which case. To let the tool work with the provided dialogue system, the designer has to set the path of the story, the picks of intensity or the curve. *ScriptTale* will then, adapt the story and decision making to follow the established rhythm. All this means that this tool can work with a dialogue to generate narrative whilst the others in the market cannot.

3. Planning

3.1. Gantt chart

3.1.1. Tasks description

The description of the tasks chronologically ordered can be found below. All of them are tagged showing the area they belong to.

Rubric 1 preparation:

Document

Prepare the contents of Rubric 1 and write them down on the document.

Research of assets:

Research

Looking for assets of the project.

Research of AI methods:

Research

Deep knowledge about the different methods that could be applied to the project.

Schema of the tool:

Programming

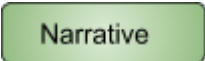
Narrative

Design the functionality of the tool, scratch an UML and make a general schema of how it will look like and how it will work.

Initial tool:

Programming

Programming the base of the tool by creating a minimum AI for the characters and the basic interface for the tool.

Dialogue system:Narrative

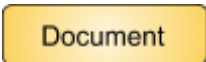
Create a basic dialogue system. Simple sentences but working with the logic needed in the project (example of the logic: combine sentences to create new ones from the features of the characters, for instance, adding tags).

Programming of the dialogue system:Programming

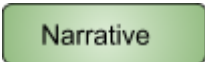
Improve the base tool by adding the methods chosen. (Possibly, programming of the methods without including dialogue system yet. That is contemplated in the following step of fixing the tool).

Test and fix the tool:ProgrammingNarrative

Detect errors, make the code cleaner, and polish the dialogue and narrative.

Rubric 2 preparation:Document

Prepare the contents of Rubric 1 and write them down on the document.

Improve dialogue:Narrative


Add more sentences and expand the existing dialogue.

Improve the GUI of the tool:ProgrammingNarrative

Improve the tool's interface, make the interface usable and the graphics appealing.

Improve AI system:Programming

Improvement of the AI methods, fix the program's errors.

Project writing:Document

End up with writing the document itself. Further stages of the project will be added during the project revision part, at the end of it.

Separate the tool from the game:Programming

Separate the game (in .dll format) from the tool (a unity package). The game should have the tool applied in it.

Decorate the tool:ProgrammingNarrative


Decide the name of the tool, create an icon, add extra features (such as sending the script of the final story to the users).

Narrative improvements:Narrative

Last final minor improvements on the dialogue system or narrative.

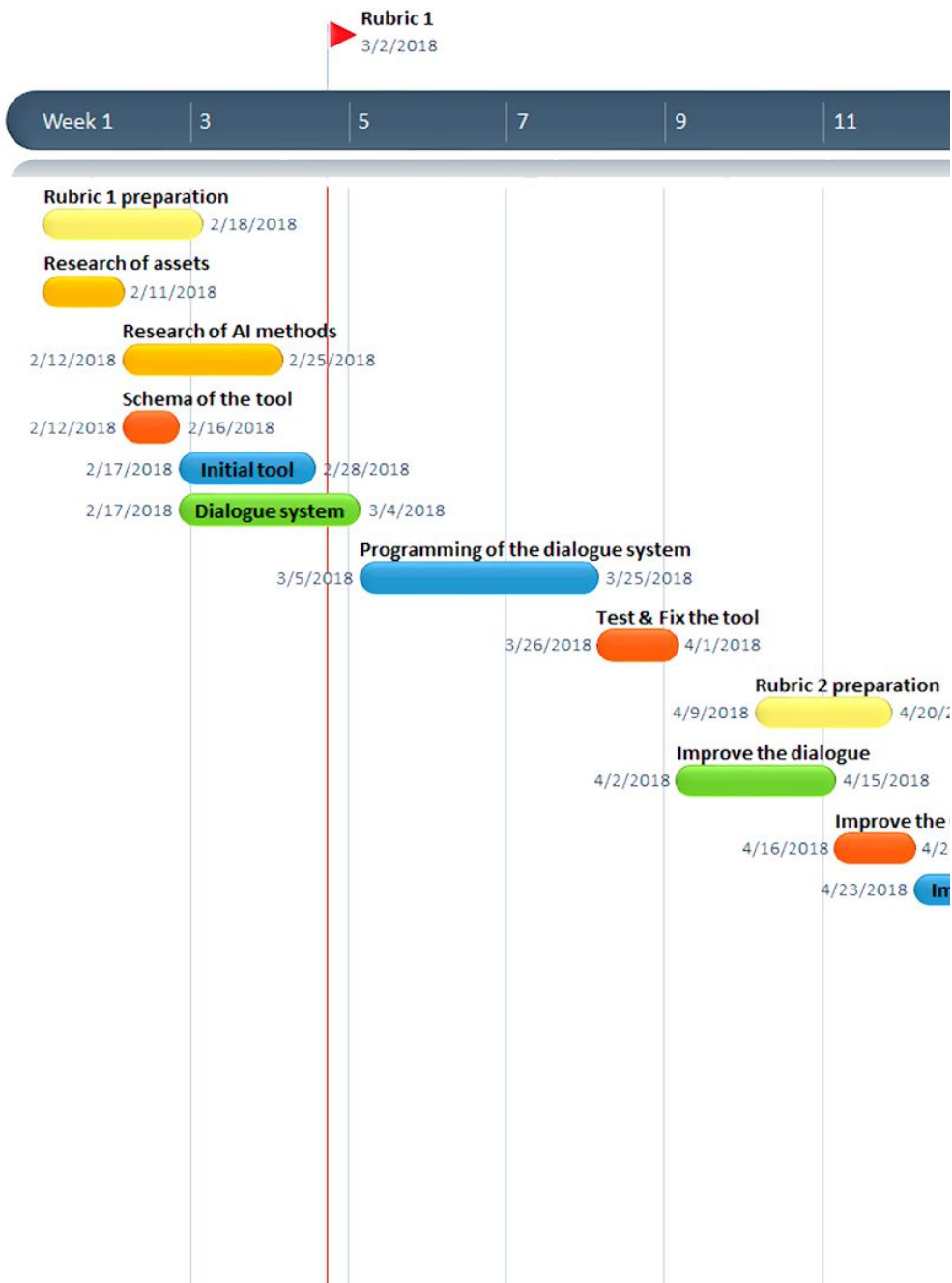
AI improvements:Programming

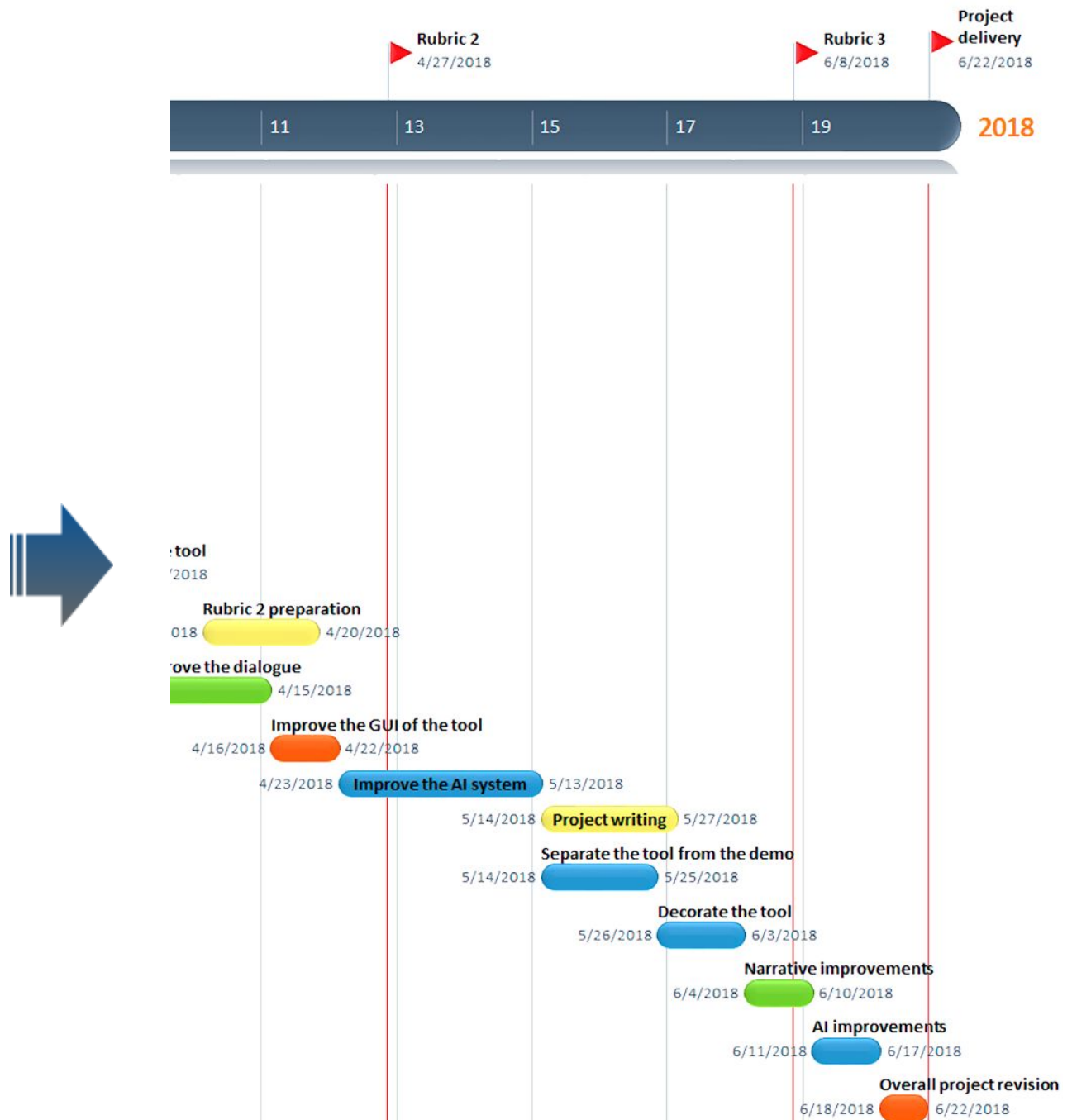
Last final minor improvements on the AI system of the tool.

Overall project revision:ProgrammingNarrativeDocument

No more programming or design to add new features will be done during these last days, only bug fixing. If a task is not finished on time, then, it will be decided where and how to cut. Include everything new on the project document and ensure the quality of the final tool.

3.1.2. Chart





3.2. SWOT

3.2.1. Analysis

Strengths:

- **Attractive offer:** The offer itself is appealing. Not only does it provide a dialogue system to reduce time and increase efficiency but also a narrative complexity higher than the one that could be achieved without the tool.
- **Innovation:** Innovation attracts the target to try something unknown if it has been proved that to work.
- **Accessible:** The target knows the place to get it and normally enters the store. If the tool is attractive enough, they can easily manage to download it and include it into a project. Target is used to it and that makes the tool accessible.
- **User friendly:** Easy to understand. There is no need of previous knowledge to use it. The tool is intuitive and most of the functionality can be applied through the UI, not through code.

Weaknesses:

- **Unrecognised brand:** There is no market behind waiting for the tool because it is a new unknown brand. That makes it complicated to get popularity.
- **Innovation:** Innovation can be both, a strength and a weakness at the same time. Being considered innovative in a field where a lot of research has been done, and is still being, is dangerous because it could mean that the idea leads to failure.
- **Oriented to very specific games:** Gameplay on games that will use the tool may offer on an open-world scenario to allow the players to interact. Cannot be applied in most of the mobile games developed by indies.

Opportunities:

- **Free entrance to the market:** Publishing the tool to Asset Store can be done for free. Moreover, it is a very popular store for the specific audience of the project. So, being shown in this place, the tool will have more visibility in the market than in other platforms.
- **No direct competitors:** There are, currently, no other tools offering this same feature to implement in games under development.
- **Time dedicated exclusively on the tool:** Most of the other developers who could think of the idea or try to implement it are working on other projects and don't have the time to work exclusively on the tool.

Threatens:

- **Less narrative in games in the indie market:** Narrative games normally developed by AAA, which is not the target of the tool. The common games that the target develops, are casual mobile games.
- **Very dynamic field of research:** The tool may not last long in the market, because innovations in AI are frequently appearing.
- **Competition against AAA games:** Despite having no direct competition in the closed market of tools in Asset store; as AAA narrative games have, normally, more visibility than the indie ones. The target could not acquire the tool, avoiding this competition with huge companies.

3.2.2. Contingency plan

All the weaknesses and threatens pointed above should be treated accordingly if they emerge at any point of the project. Thereupon, the way of coping each of them is explained.

- **Unrecognised brand:** Work on the marketing before launching the tool to test the interest of the target on the product. The tool could then be fixed to fit a community that should be preserved until the launch, having by that time a minimum amount of buyers ensured.
- **Innovation:** Test the product in the market at the very first moment a Minimum Valuable Product is achieved. If at that point, the tool does not seem to work, future economic losses can be mitigated.
- **Oriented to very specific games:** This could be driven to a positive feature if a marketing plan is correctly developed and focused on the appropriate channels, the tool could reach the specific target that is eager to develop this specific type of games.
- **Less narrative in games in the indie market:** To face the possible consequences of this scenario, instructions of use will be provided. Showing the ease of use of *ScripTale*. It could be proved that this feature can be implemented in Indie games.
- **Very dynamic field of research:** Frequent updates and research in the field to avoid becoming obsolete.
- **Competition against AAA games:** In case the independent market seems to be afraid of getting the tool and competing with bigger narrative game, try to open the field and reach these AAA. The tool could be adapted to them with all the detailed information of the development is shared to let them implement something similar on their own engine.

3.3. Costs analysis

3.3.1 Costs overview

Work packages

Work Package	Title	Type	Leader
WP 1	Research of assets	Research	Developer
WP 2	Research of AI methods	Research	Developer
WP 3	Schema of the tool	Design & Programming	Developer
WP 4	Initial tool programming	Programming	Developer
WP 5	Dialogue System	Narrative	Screenwriter
WP 6	Programming the Dialogue System	Programming	Developer
WP 7	Test & Fix	Programming & Narrative	Developer
WP 8	Improve the dialogue	Narrative	Screenwriter
WP 9	Improve the GUI	Programming & Narrative	Screenwriter
WP 10	Improve AI System	Programming	Developer
WP 11	Separate the tool from the game	Programming	Developer

WP 12	Decorate the tool	Design & Programming	Developer
-------	-------------------	----------------------	-----------

Table 3.3.1. a Work Packages

Partners

Partners	Type of job	Title	Average monthly cost
Developer	Part-time	Developer	1600 €
Screenwriter	Part-time	Screenwriter	1100 €
Graphic designer	Outsourced	Graphic designer	-

Table 3.3.1. b Partners

Budget

Work Packages	Costs categories				Total budget
	Human resources	Software	Hardware (Amortisation)	Indirect	
WP 1	200 €	24 €	5 €	626.29 €	855.29 €
WP 2	600 €	0 €	5 €	626.29 €	1231.29 €
WP 3	800 €	0 €	5 €	626.29 €	1431.29 €
WP 4	1200 €	56.6 €	5 €	626.29 €	1887.89 €
WP 5	1600 €	56.6 €	5 €	626.29 €	2287.89 €
WP 6	1600 €	56.6 €	5 €	626.29 €	2287.89 €

WP 7	800 €	56.6 €	5 €	626.29 €	1487.89 €
WP 8	1600 €	56.6 €	5 €	626.29 €	2287.89 €
WP 9	400 €	56.6 €	5 €	626.29 €	1087.89 €
WP 10	1200 €	56.6 €	5 €	626.29 €	1887.89 €
WP 11	400 €	56.6 €	5 €	626.29 €	1087.89 €
WP 12	900 €	56.6 €	5 €	626.29 €	1587.89 €
Total	11300 €	533 €	60 €	7515.52 €	19408.52 €

Table 3.3.1. c Budget

Estimated deviations

A deviation of 10% should be estimated for human resources as the time of work required could fluctuate and vary each week. The project being a research type of study, the amount of hours to be invested is difficult to predict, the evolution will depend on all the results obtained through the development.

On the other hand, indirect costs can be forecasted by current references in the market. But, as the figures can change and this depends on external factors, a lower deviation must be considered, a 5%, for example.

3.3.2. Costs breakdown

Human resources:

This project is developed individually so human resources would only include one manmonth. However, some of the tasks in this project could be outsourced, or even split into different roles. As a consequence, the cost analysis of human resources has to take it into account.

Role	Price per hour	Amount of hours	Total cost
Developer	20 €/h	320 h	6400 €
Screenwriter	20 €/h	220 h	4400 €
Graphic designer	25 €/h	20 h	500 €
TOTAL	-	560 h	11300 €

Table 3.3.2. a Human resources

- **Assumptions:** As the Graphic designer's tasks don't require a high implication in this project, it is assumed that a freelance designer will be required for some hours. The salary of the designer is then the average price that would be asked by a freelance. Developer and screenwriter are meant to be into the project for partial-time, that's why their price per hour looks more economic.

Software

Software	Price per unit	Units	Total cost
Unity Plus license	360 €/unit	1 unit	360 €
Microsoft Office	149 €/unit	1 unit	149 €
Unity assets (Characters & Environments)	12 €/unit	2 units	24 €
TOTAL	-	-	533 €

Table 3.3.2. b Software

- **Assumptions:** The developers acquire a license for 1 year and 1 account. Unity assets bought shall be low price assets but not some of the free ones. Microsoft Office offers a license for 3 years, so it may be used in further projects of the studio.

Hardware

Hardware	Price per unit	Unit	Lifetime	Total cost	Amortisation
PCs	900 €/h	1 unit	5 years	900 €	60 €
TOTAL	-	-	-	900 €	60 €

Table 3.3.2. c Hardware

Indirect costs

Type	Price per month	Amount of months	Total cost
Office rental	1200 €/unit	4	4800 €
Light, water & other supplies	150 €/unit	4	600 €
Self-employed quota	264.44/month and person	4	2115.52 €
TOTAL	-	-	7515.52 €

Table 3.3.2. d Indirect costs

- **Assumptions:** *As the proper size of a small office is less than 100 m², it is assumed the property will sit on a 90 m² surface, located in Barcelona. Self-employed quota must be paid twice, having the developer and the screenwriter as the main share-holders.*

3.4. Management tools

Hacknplan and **Trello** will both be used to keep the track of the development. Hacknplan tables may change for each milestone or in a short settled period. Trello, meanwhile, is thought to be used to store a more general picture of the project, in other words, all the project tasks will be specified in Trello as generic capsules that could embrace multiple specific assignments.

4. Methodology

In the videogames industry, it is common to use a pre-production, production and post-production model-based methodology. However, this study does not consist on a videogame itself but on a tool, therefore, this methodology will not work efficiently on a project of this kind.

An agile interactive method would be the most suitable to develop the project. The different stages could change dynamically as it consists of a research that has to be iterated, tested, and change continuously. Scrum is one of these methodologies that would perfectly fit the project's typology, so that is the one to be used.

Scrum, being based on sprints, can be easily managed with such a management tool like Hacknplan.

Weekly stages:

- **Beginning of the week:** A backlog update will be done and the tasks to be developed over a week will be noted down.
- **During the week:** Daily sprint review of the tasks done, balancing remaining time/tasks and estimations established to get to know one own's error rate.
- **End of the week:** Try to close all the tasks. If that is not possible, draw to a close by deciding which features are to be cut given that the time is limited.

4.1. Tracking of the project

Hacknplan, as used for the management, will be accessible at any time of the project and it will also show its progress, current, and past.

As happens with Hacknplan, **GitHub** can also provide a good tool for monitoring the project. Through all the commits, if described properly, it is possible to see the advance.

However GitBash will be used to commit all the project's changes instead of GitHub, tool which also allows this action, through GitHub will be used to check the progress. Github provides an intuitive User Interface for this monitoring feature.

4.2. Validation tools

To check the authenticity of the theoric information collected through internet, consulting to a professional in this field will be done. **E-mailing** the Doctor in Artificial Intelligence Chris Martens could be considered by itself a validation tool for the investigation part.

Before testing, because of using Unity as an engine, a suitable debug needs to be done. Linking Unity to the IDE **Visual Studio** makes it possible not only to implement the code but also to check it at the same instant, to debug it.

Visual studio will also be used to test, but not as an exclusive tool, because testing needs to be done by using the program and as such **Unity Editor** will meet this needs.

After having fixed the bugs found through this tool, another testing round will be done by building the application. A testing scene will be creating during the testing session to evaluate its effectivity in runtime, not only within the editors, but also the AI scripts executing in game.

Validation is also required for the narrative aspect of the project so, in this case, online tools will be used, for instance: **correctors**, **synonym dictionaries** and **dictionaries**.

5. Development

The development process starts by conceiving the structure of the tool, deciding which parameters to obtain by the input of a designer or which ones should be left up to the tool thus the planning of further tasks gets easier.

However, this led to structural problems on how many variables should be handled by the designer and once decided and how an usable UI could be displayed leading to the ease in managing and editing a vast number of parameters.

For these reason, parameters to be handled have been divided into three different fields: the relationships' editor, the curve editor, the states' editor. These editors, however, need other structures to get to work, such as the characters with its emotions and behaviours or the messages classified from the dialogue.

Before the implementation, all of these had to be clearly designed to avoid inconsistencies or negligible work.

5.1. Design of the tool

After seeing other concepts of Procedural Narrative that can be found in the State of Art section, letting the characters plot the story by their actions was the way to start with the tool. However, with the new idea of including a dialogue to fit the narrative, the Linear Logic method used in these other projects was not effective enough and this other methods to be combined with were required in the *ScripTale*'s implementation.

To behave in a way or another, the characters will be driven by emotions and each consequent state after an action should affect some characters in some way. So, the emotional component is added now to the linear logic relatives states.

At this point, the necessity of including a grade of importance was needed on how the characters would be affected. Consequently, the tool should include weights. All the interactions with the characters would affect them in some way so, all the inputs

are designed to control emotions, and emotions will drive the story. States (results of the actions performed) and messages (what players say) have emotional implication.

This emotional implication is translated into a natural behaviour of the characters in taking actions. To sum up, in this tool, actions are still plotting the story but, being affected by the emotions resultant of the original characters' behaviour and other characters' actions, so as to get a fitting dialogue in all this procedural generation of narrative.

The artificial intelligence of the tool works with linear logic and weights.

- AI concepts:

Working with weights means implementing a **Goal Oriented Action Planning** (GOAP). It is a system that allows the agents (characters) to plan a sequence of actions to satisfy an aim. This sequence of actions depends not only to the goal but also on the current state of the agent.

Each possible action has multiple weights of necessities it will satisfy. However, the actions available can present a condition. That is the point where the current project combines **Linear Logic** (LL) with GOAP.

LL will ensure that an action cannot be done until another one has ended, an item has been consumed or, on the other hand, some actions can happen simultaneously. Linear Logic states the conditions and the end point of each action.

ScripTale implements LL to decide the available spectrum of actions from the current states, the possible ones and the demand of the intensity curve. GOAP is used after to get the most likely action or message to occur given the character current state and the available actions filtered by the Linear Logic step.

- Initial design schemas to start implementing:

All the elements in the schemas were added and modified by the schemas' own nature if they had to follow the two rules of: emotional implication for the decision making and actions to plot the story.

Base Classes

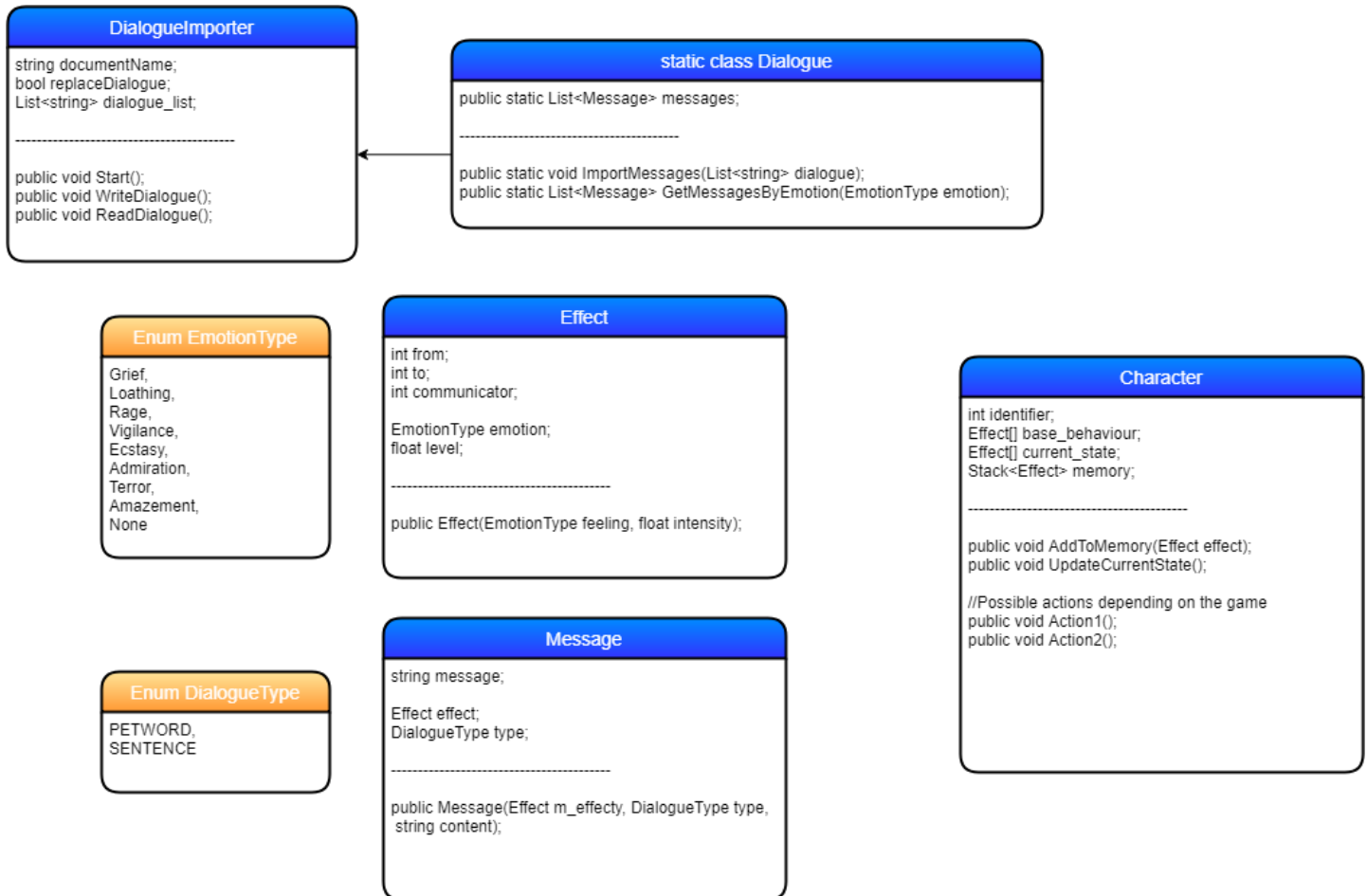


Fig. 5.1.1. Design schema of the base classes of *Scriptale*.

Editors

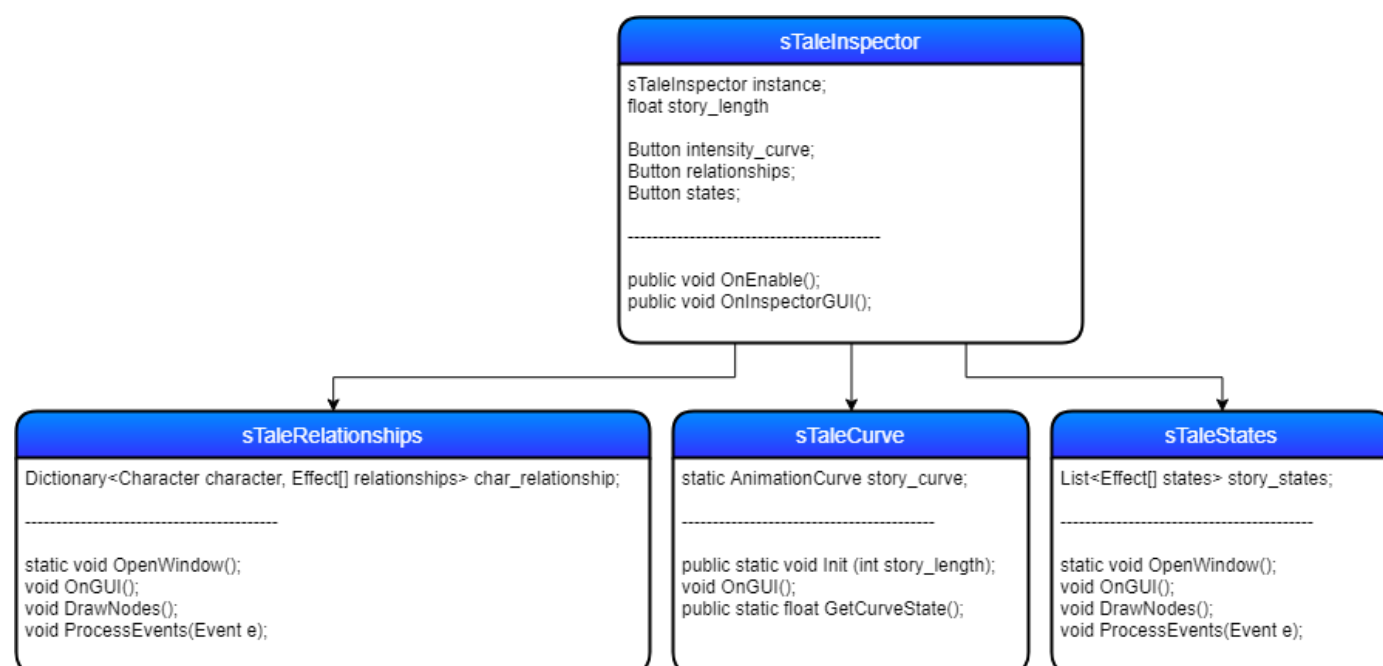


Fig. 5.1.2. Design schema of the editors of *ScripTale*.

5.2. *ScripTale* Initial implementation

5.2.1. Curve Editor

The curve editor provided to the designer is meant to be driving the story through an intensity path desired. It would allow the designer to control the time the climax will happen or the more relaxing parts of the story.

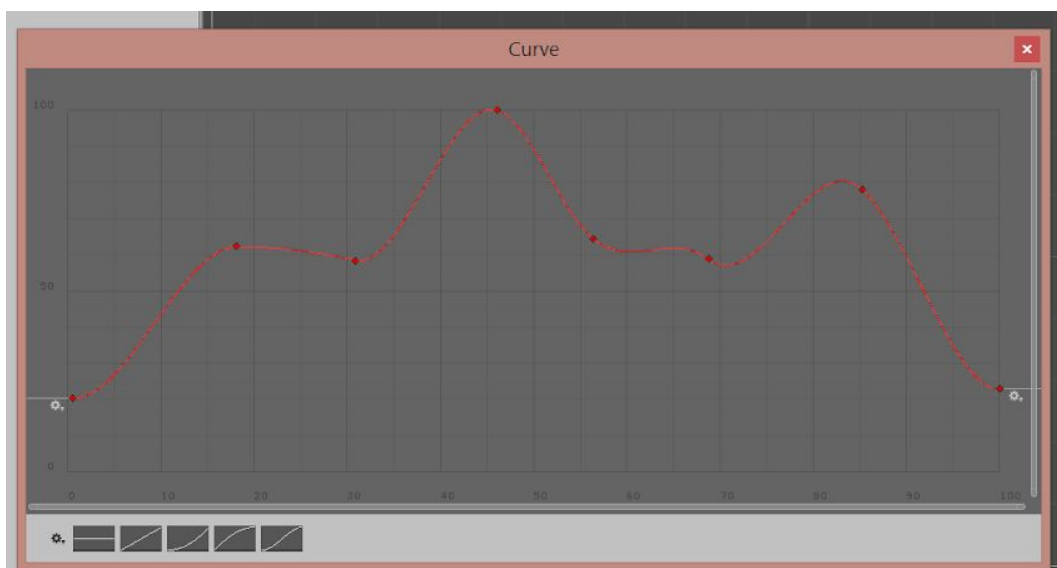


Fig. 5.2.1.a. Curve editor of *ScripTale*.

- Problems:

The curve editor actually did not suppose any striking problem. The only trouble that was generated had been when *ScripTale* was applied to a game consisting of narrative parts (interactive parts where developing the plot of the story) and gameplay parts, the curvature was still advancing in time. The problem was easily solved by adding an option to pause the curve by calling a function, if the narrative needs to be stopped, and resume it when required.

However, despite the tool is thought to be used by designers, this lead to the simplest as well as major inconvenient of *ScripTale*. It is the programmer, not the designer who has to take the already mentioned problem into account. The solution

is easy but the call to the following functions in code is required to make it work. It is not a matter of design. These functions correspond to the ones shown in the following picture:

```
public static void PauseCurve()
{
    time_paused = Time.time;
    curve_paused = true;
}

public static void ResumeCurve()
{
    curve_paused = false;
}
```

Fig. 5.2.1.b. Curve functions of *ScripTale* necessary for the programmers.

Another trouble was matching the duration of the curve with the last state of the story. This has not been solved yet, but a further implementation with improvements on this could be a possibility for an expansion of the current tool, *ScripTale*, in subsequent research.

5.2.2. States Editor

The output of this editor is the second to evaluate. It consists on a tree of states, one state can lead to multiple others. Parallel states could never occur. The tree begins with an initial state and expands to lots of possible ends. The tree is being navigated through the actions of the players. In the example of *The Hare and the turtle*, the final state of the turtle winning the race would be a consequence of the continuous running of the turtle and the decision of laying down of the hare. So, from the state “*Race in progress*” to the state “*Race won by the turtle*”, both these actions must have happened.

Actions available depend on the ones that have already been consumed or passed by through the progress in the story. The current state of a character won't be changed until the action in progress has been consumed or, in other words, has ended.

To sum up, from one state to another at least one action has to be done. An action can be performed by the characters themselves if the intensity curve is asking for a certain state to happen, or by the player.

In other words, if the story-line plotted by the designer in the intensity curve is meant to have a high intensity, the following state will be one with a high intensity, a state having a huge emotional implication on a character or more than one. The higher the intensity, the high is the importance of the emotions encapsulated in the state to be chosen.

States, as messages do, contain an array of characters affected and the levels in which their emotions are modified.

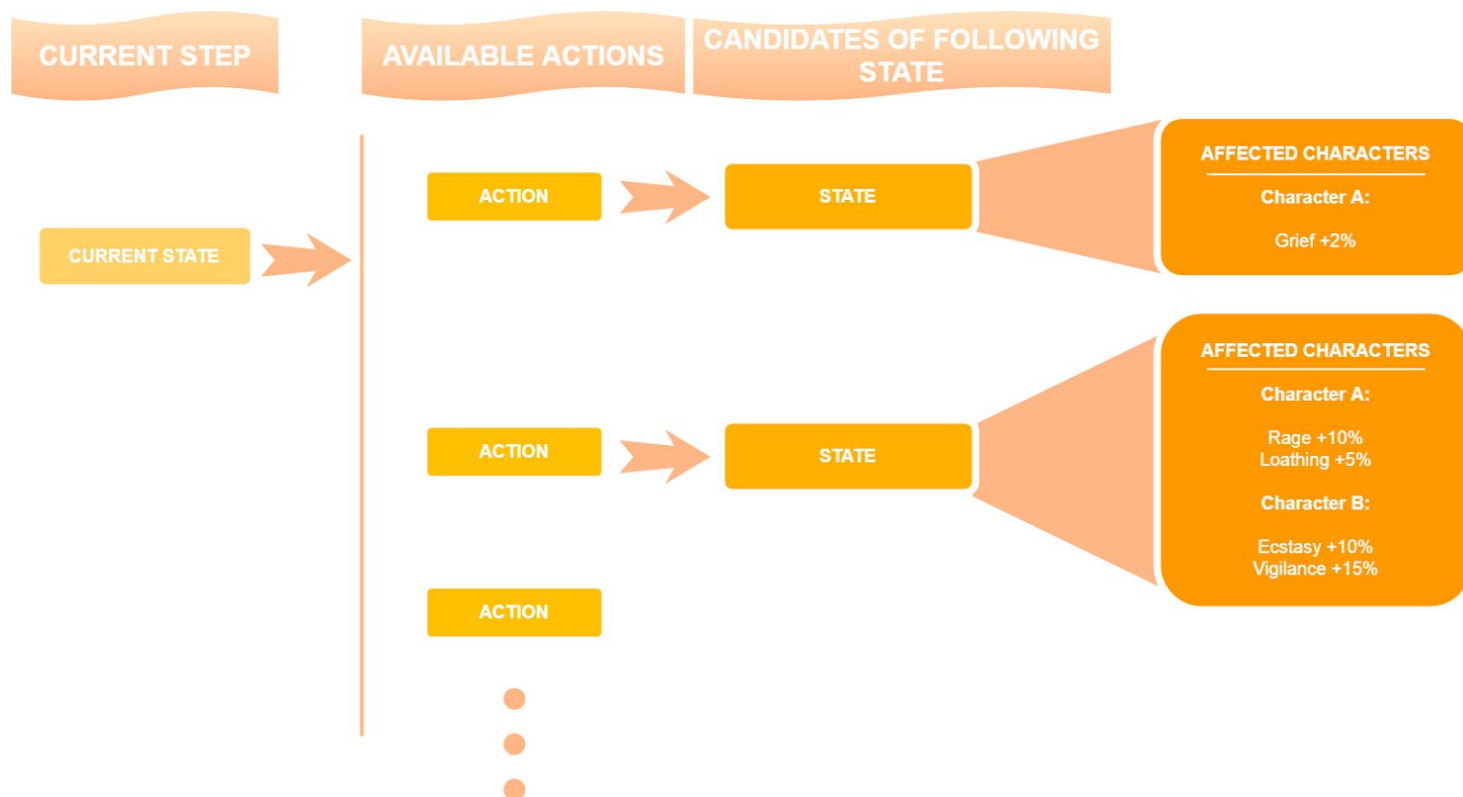


Fig. 5.2.2. Schema of the States functionality

After evaluating the states offered at a certain state, the tool takes the output of the intensity curve and depending on its value will follow a path or another. If the current intensity of the curve is equal to 80%, the tool will take the action with an implication of 80% or close to it.

- Problems:

The story-line, the narrative generated, is driven by the actions of the characters, the states in which they are found after the characters' input. This is what makes narrative procedural, the characters plotting their own story with their behaviour. Other implementations give the absolute control of the story to these states and actions.

But, in this tool, as having a dialogue system behind, this has to work accordingly to other parameters that would control the dialogue. Controlling a dialogue system

could not be carried out by only having an editor with the desired states, the current states and the possible actions was not enough to control a dialogue system. So, this was the first and the most important problem which the states' editor led to, the **necessity of other editors** to control the narrative. If only having a this input of the designer, the dialogue wouldn't have been accurate.

Parameters derived from this editor, then, must be linked to the storyline and the type of character behaviour, the output of this editor must have a clear relevance in the dialogue, combined with the other editors.

5.2.3. Relationships Editor

The relationships' editor is a node editor to connect characters. Each line linking two characters is a relationship. This one connects characters and how each one modifies the action or message the other receives. For instance, if a character emits a message that affects negatively to an other to whom it gets along, the effect of the message will be softened.

Each relationship will have, as a result, an effect not to a character but to a message or action that is addressed in the same direction of the relationship. The example below illustrates the performance of a relationship.

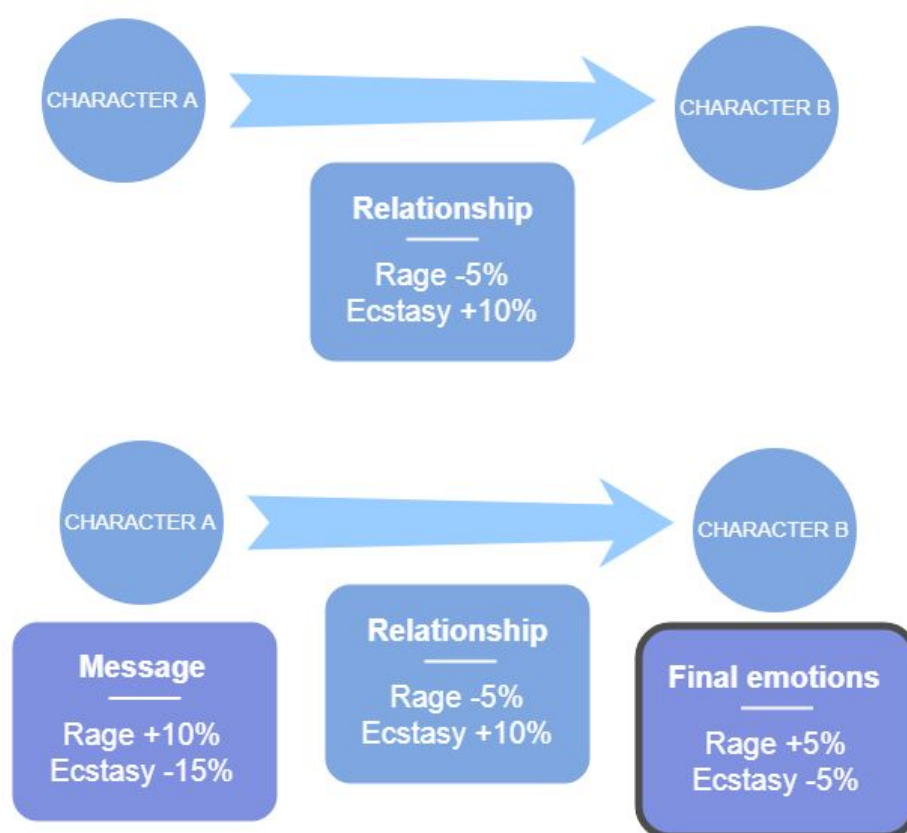


Fig. 5.2.3. Schema of the Relationships functionality

- Problems:

In spite of the fact that the Relationships' editor consists on mathematic modifications on the weight of emotional implication of an action or message, it was initially static. It means that the relationship does not evolve along the story. Some characters could hurt badly other characters and they could still get along after that and repeatedly mistreat each other without many consequences. Each character should have a memory of messages and actions received and modify this relationship dynamically beyond the game, accordingly.

5.2.4. Dialogue System

The script is downloaded from Google Drive and refreshed at everytime the project is played if the designer allows it checking the toggle box in the dialogue system prefab (marked by default).

This way, *ScripTale*, makes it intuitive and usable to edit the dialogue in a such a comfortable and useful tool as Drive documents. From a drive document that can be modified at anytime by any member of the team in a .txt format. If the designer follows the writing rules specified in the README.md file, the dialogue system should be passing all the parameters to the game and this one will always have a quick access to the dialogue at any time.

- Problems:

The editors of *ScripTale*, could be implemented by using the editor features that Unity provides. However, the dialogue system caused other complications. As the dialogue system depended on third parties to write the documents, if these documents had to be read in the script, those programs may ask for the computer to have a license to use the calls in the API. It could be managed to get the license in the computer in which the tool was developed but, a local copy of the dlls that provided the API was not allowed. That means that the dlls won't be copied to other computers if they don't have the license.

To break the limitations of this kind of software, the Dialogue System was changed to work, as it currently does, through Google Drive.

The Asset Store of Unity and even other platforms on the Internet, provide Plugins to integrate the use of spreadsheet Excel in a project. However, ScripTale works with document files, not Excel, and, as it was complicated to find a Plugin with this functionality, this one was implemented in the code of the tool itself through the Google Drive API without any other Plugin. The **research of Plugins** and the final decision of **coding the desired functionality** were the two main problems that appeared after solving the precedent conflict relative to **software licenses**.

5.2.5. Characters

Characters are entities with an id different one to each other to be quickly identified. They will also own a stack of events, each event consisting on a structure with a pointer to another character and an array of emotions.

The stack pretends to simulate a memory, it keeps track of what has happened to the character and what could break or modify a relationship.

Finally and the most important, the character is constituted by two sets of emotions. The original ones that define it and determine its way of acting and behaviour. If a character tends to be aggressive by nature (high level of rage emotion by default), for instance, a message transmitting rage will increase the rage current state of the character.

- **Problems:**

Characters must have a memory to keep track of what has happened. However, the knowledge of the events is not taken in consideration. These caused two problems:

- Messages did not have any effect on them at any moment if they are not applied to the character.

- The character is affected even if it doesn't know the source of its emotions because has not received directly the message or the action.

A temporary solution is not to apply an effect of a message that has not been received directly by the character and apply the effect if it gets to know it. It is complicated, though, to track if it discovers a message that is already in its memory stack.

5.2.6. Messages and actions

Messages and actions are actually the same thing. Both of them consist on structures containing the id of the character emitter, the one of the receiver and a dictionary with the ids of the characters affected as keys and the array of emotions that the message or the actions produced.

The difference between messages and actions is the output. Messages are meant to pick a type of dialogue depending on the emotions of the emitter character and to create an effect on the receiver. This is, then, the output of the message. However, an action derives to an state. It is also chosen by the emotional state of the character, however, in this case, it takes into account the conditions of the story.

To quickly pick up a message from all the loaded ones, these are the helper functions called by *Scriptale* when building the dialogue.

```
public static List<Message> GetMessagesByEmotion(EmotionType emotion) {  
  
    return messages.FindAll(m => m.effect.emmotion == emotion);  
  
}
```

```
public static List<Message> GetMessagesByEmotionAndLevel(EmotionType emotion, float level, float threshold) {  
  
    return messages.FindAll(m => m.effect.emotion == emotion && (m.effect.level <= (level + threshold) &&  
        m.effect.level >= (level - threshold)));  
  
}
```

- Problems:

Although actions and messages are conceptually the same type of structures, messages are not supposed to be able to get to a state and so, it could be difficult to reach to a state. Some possible solutions were or choosing an action right after a dialogue happens, which would still feel antinatural to cut a conversation and start an action immediately; another one could be to start an action depending on the resultant states after a conversation. This last one is the approach taken in this tool, despite there are probably better implementations to solve it.

5.2.7. Emotions

As it has been stated multiple times in the current document, characters, messages, actions and states are based on emotions. In a more technical aspect emotions consist on structures which simply contain the enum type of emotion and a float value corresponding to the level.

The basic emotions to define a character are:

```
public enum EmotionType
{
    Grief,
    Loathing,
    Rage,
    Vigilance,
    Ecstasy,
    Admiration,
    Terror,
    Amazing,
    None
}
```

All of them have been chosen to achieve a human-like realistic behaviour and different archetypes of characters by mixing various levels of emotions. On the basis of that postulate, the feelings that have been applied in the current project actually follow the ones that are thought to constitute the human being. Plutchik R. (1927 - 2006) had developed a wheel of emotions to describe the people with feelings. This has been the inspiration for the EmotionType choice of options.

- Problems:

Before realizing some research on psychological papers was needed to develop a variate and spectrum of emotions to define a character, these basic feelings where needed. However it was complicated to pick up the right choice of emotions that

could be combined to get different behaviours without any psychological knowledge.

The real matter about the inclusion of emotions in the project was the unexpectancy of the task itself. It was not taken into consideration at the time of the planning. However, due to the nature of research of this project, these emergent situations were thought to occur so, some generic tasks include in themselves investigation and implementation. The Gantt chart can, then, remain the same, as “Emotions” can be considered a task inside the “*Dialogue system*” one.

5.3. Initial workflow of *ScripTale*

The big picture of *ScripTale* would look like this schema.

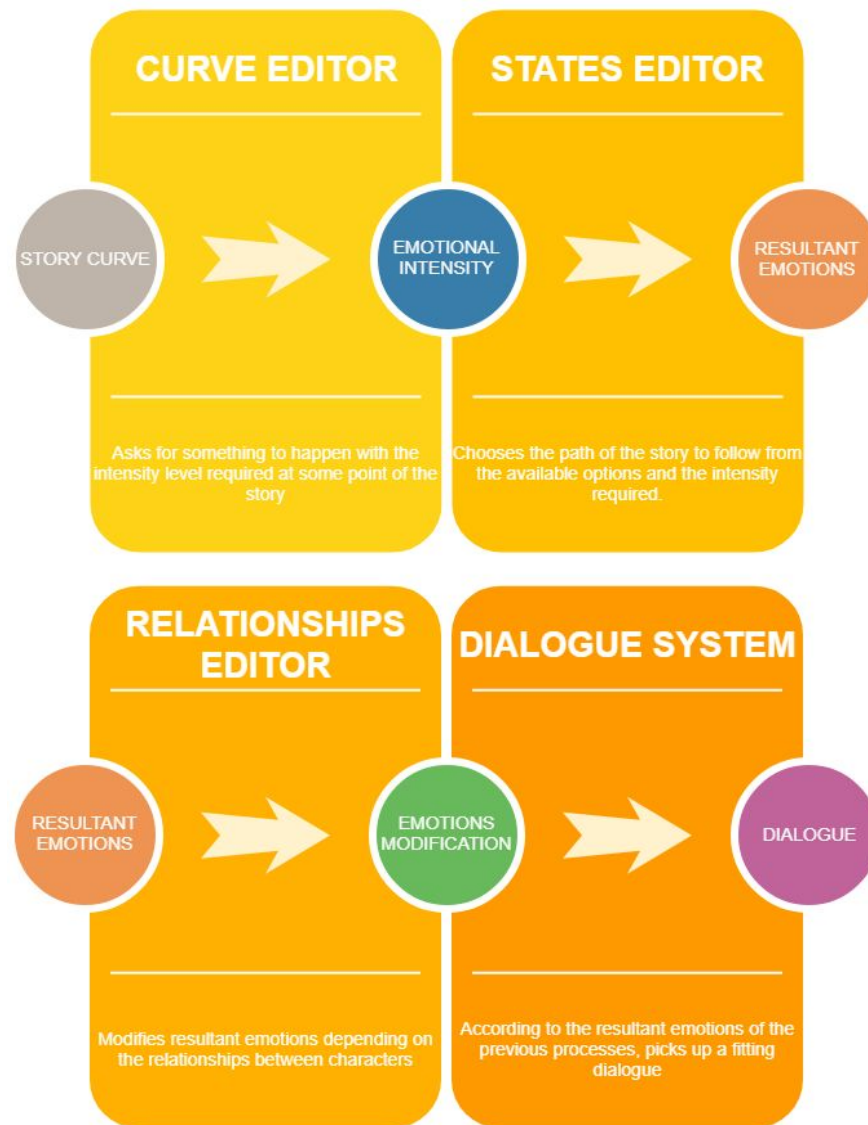


Fig. 5.3. Workflow of *ScripTale*.

5.4. Last state of *ScripTale*

5.4.1. Curve editor

The tool has definitely removed the Curve editor because of design incongruities. First of all, usage of the tool gets more complicated as more elements are being added. Moreover, *sTaleCharacters* have currently been totally programmed and have a huge impact on design, so more items to control were being developed. What is more, story had very little influence in the story and its perks were fewer than the problems it caused. It was only taken into account to get the next possible state, depending on the intensity on time.

The corresponding function in which the curve was involved was the following. Were the tool just got the intensity directrices from it and combined it with the intensity requirements of each of the states. Then it took the most suitable state for the established curve.

```
private static void GetNextState() {
    float intensity = sTaleCurve.GetCurveState();

    float[] level = new float[current_state.children.Count];

    for (int i = 0; i < current_state.children.Count; ++i) {
        float average = 0;

        int count = 0;

        for (int e = 0; e < current_state.children[i].content.returnEffect.Count; ++e) {
            for (int c = 0; c < current_state.children[i].content.returnEffect[e].conditions.Count; ++c) {
                average += current_state.children[i].content.returnEffect[e].conditions[c].level;

                count++;
            }
        }
    }
}
```

```

        average = average / count;

        level[i] = Mathf.Clamp(current_state.children[i].content.conditionsSet * average, 0.0f, 100.0f);
    }

    int following_index = 0;

    for(int lev = 0; lev < level.Length; ++lev) {

        if(Mathf.Abs(intensity - level[lev]) < Mathf.Abs(intensity - level[following_index])) {

            following_index = lev;

        }

    }

    following_state = current_state.children[following_index];
}

```

The states Editor is a way itself to control the intensity depending on the conditions set by the designer and no further editors should be needed to accomplish this aim.

5.4.2. States Editor

The States Editor functionality remained almost the same as what was initially designed and implemented. As the decision of removing the intensity curve was removed, the next state to follow in the tree had to be decided in a self-contained way. Without the input of the designer to plot the path, the tool had to do it by itself, so GOAP takes more relevance in this process.

Various goals were planned for each state. From the current state, the story has different aims. To decide which one to take, the tool checks the conditions in each of these states and calculates the distance between the current emotional level of the character and the level set in the condition. The one to be choose is the most likely to happen. In technical terms, the cheapest option, the one that would take less iterations.

5.4.3. Relationships Editor

This editor remains the same as was planned. When the editor is first opened, all the characters appearing in the story have to be already put in the scene. All of them will appear as a node and when they are connected to each other, their relationship has to be set. The relationships can be saved in an own extension file and this file or other files derived from the editor can be assigned to the *ScripTale* prefab to be used over the course of the game.

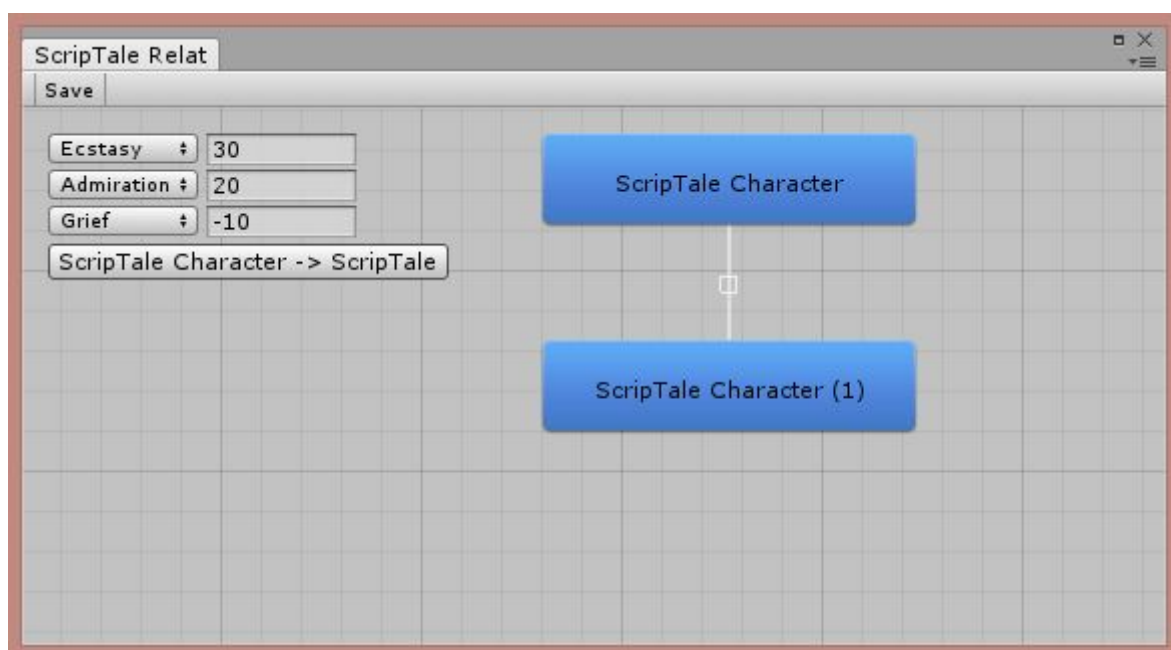


Fig. 5.4.3. Relationships editor of *ScripTale*

5.4.4. Dialogue System

Few changes have been applied to the Dialogue System. The main workflow through Google Drive has been kept: The document must be uploaded in Google Drive and it will be downloaded directly throughout Unity, without any need to download it in the computer and passing it to Unity afterwards.

The principal and only improvement is that it is the user who chooses when to download the dialogue file by checking the toggle provided in the Dialogue Importer prefab of *ScripTale*. The document is not refreshed automatically but only when the designer wants to and selects the option to download it again.

All the instructions of use are specified in the *Readme.md*.

5.4.4. Characters

Characters' narrative behaviour is all found in the script *sTaleCharacter*, that has to be attached to the Character prefab in the scene. It is one of the most complex script in *ScripTale* because it consists in an agent that interacts with the editors to act or talk depending on the current state and also on the messages and actions that the character itself owns.

These are all the variables that the character needs or controls:

```
[SerializeField] public int identifier;  
  
[SerializeField] public Effect[] baseBehaviour;  
  
[SerializeField] public Message[] characterMessages;  
  
public Effect[] currentState;  
  
List<Effect> memory;
```

```
[SerializeField] TalkEvent talkAction = null;  
  
[SerializeField] List<AvailableAction> availableActions;  
  
[SerializeField] float actionLastingTime;  
  
[SerializeField] InteractEvent conditionToInteract = null;  
  
[SerializeField] int affectsRelationship = 2;
```

identifier:

Number to identificate every character. It has to be assigned by the designer and cannot be repeated in different characters. This id number is used to look up for a character in a most efficient and quick way than making a search by the name if the occasion can allow it.

baseBehaviour:

Original mood of the character. It is the starting point to modify its mood over the story.

characterMessages:

These are all the messages that can be said by a character. They have to be inserted by the designer and classify by category. Messages contain a string, the text that will appear on making a character talk. Each of them will cause an effect, an emotion with its linked intensity to a character. All of this is supplied by the *sTale* prefab to be specified on it.

currentState:

This variable is quite similar to the *baseBehaviour* as its aim is to keep track on the modifications applied to the character. It shows the current mood of the character and it is handled by *ScripTale* as the story goes on, not by the designer.

memory:

The memory of the character is used to keep updating the relationship between characters depending on the messages received from the other character interacting with it. It is up to the tool to fill this list.

affectsRelationship:

Variable strongly related to the memory. The messages will affect to the relationship from the number of messages received that can affect relationship. In other words, a relationship is not being changed at every single interaction but at a certain number of interactions that can affect it; this number of interactions correspond to this variable: *affectsRelationship*. It is set by default, however, it can be changed by the designer in the inspector of *sTale*.

talkAction:

It is the action on talking. It refers on an action that the character could do on talking, for instance, an animation, but not a more complex one. It is assigned by the designer as it will depend on the particular script of each game to control the character.

availableActions:

These are all the actions the character can perform in the game that the designer wants the AI to take control of them. Some of them could be to Hit, to Jump, to Run... MonoBehaviour functions that are called on character's interaction in function of the intensity of the message.

actionLastingTime:

To control the pacing of the story and avoid instantaneous responses, the *actionLastingTime* controls the time it takes this character to act. It can't interact again until the time has come to its end. It is done to avoid a very quick narrative that can't even be noticed. Time in this variable has to be specified in seconds, as it

will be called by a coroutine to wait until the character can go to the idle state after acting.

conditionToInteract:

Characters will only interact with the others if the condition of interaction is accomplished. This condition may vary from a game to the other, as well as the actions that characters can develop, so it is up to the user to assign in the inspector this condition function or the available actions of each character from their own MonoBehaviour script attached to the same Character GameObject.

The user has to be careful, though to send the sTaleCharacter to interact implementing a function which sends the object of the character who wants to interact with the other.

```
bool GetCondition(GameObject other) {  
  
    MethodInfo methodInfo =  
    UnityEventBase.GetValidMethodInfo(conditionToInteract.GetPersistentTarget(0),  
    conditionToInteract.GetPersistentMethodName(0), new System.Type[] { typeof(GameObject) });  
  
    return can_interact;  
}
```

5.4.5. Messages and Actions

Messages and actions made a substantial shift from the first implementation. At first they were thought to be global and parallel to execute. Messages and actions could never happen at the same time.

In spite of this initial idea of design, messages and actions ended up occurring at the same moment. This has been brought up by the necessity to perform an action and show a message on interacting, differentiating the actions due to narrative from the ones that the game character will be doing in its separate controller.

Messages and actions basically still mean effects of emotions at levels that apply to a character but, they are now managed by the characters themselves acting as agents of *ScripTale* and not by the more generic AI derived from the editors in the *sTale* AI controller. This decision was taken after realizing some messages should be of each character instead of making the access to say the messages or perform a certain action, global to all the agents in the game.

5.4.6. Emotions

As the emotions were the pillar of the current tool, they were not modified at any time. When the idea of the eight basic emotions came up, these were established as fixed in the workflow of the development.

To sum up, the emotions are combined with an intensity level clamped between 0 and 100 to have an effect to a character.

5.5. Final workflow of *ScripTale*

A final summary of the workflow of the tool would be like the following picture:

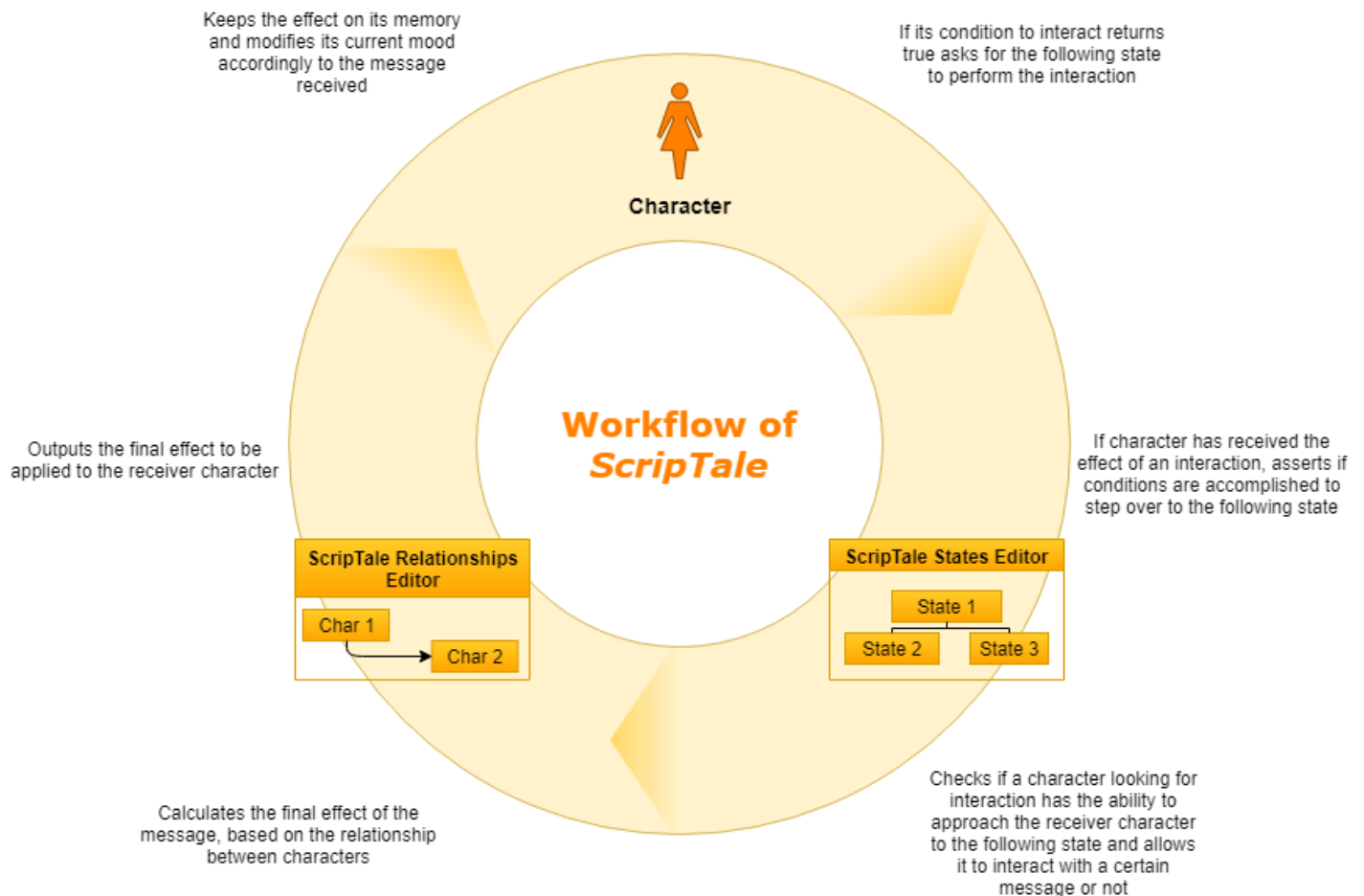


Fig. 5.5. Current workflow of *ScripTale*

6. Conclusions

Beyond the development of the project, the tool had to be fitting the needs that the objectives requiring. The main aim of spontaneity in the story that was settled as a pillar that could not be moved aside, as is the challenge of the current subject of study. Consequently, the process should keep adapting to always follow this ambitious milestone.

During the research in the State of the Art, a similar project was found in Practices for Procedural Narrative Generation (Martens Chris, 2017). The given advice of characters' behavior plotting the story truly inspired the design of *ScripTale*. However, this narrative telling, did have nothing to do with the dialogue of the story itself and its possibilities were still finite, with paths entirely under the designer control. As it is written in their own tool's write: "TeLLer: an interactive linear logic prover and proof explorer that is currently being used to study causality in interactive story telling".

Despite the Linear Logic approach was kept, to enquire further and make a procedural story with also a dialogue generating on the way, this one could not be the only solution. The problem comes at this point, with this being not the only solution. So, to reach the goal of the tool, as was finally achieved, more than one solutions were needed.

The first solution came to be the States Editor with a tree structure that tried to emulate that Linear Logic behavior of *TeLLer*. To sum up, it would be meant to be a behavior tree with much less conditions. Parallel nodes cannot occur simultaneously and a condition has to be accomplished to step into the following node.

This other tool does something similar to the States Editor of *ScripTale* but with an elegant and clean syntax, to be used by coders, not by designer. So, the problem is solved equally with two different implementations due to the difference in target.

```

1 % Narrative resources
2 convent : type.
3 education : type.
4 grace : type.
5 novel : type.
6 ball : type.
7 escapism : type.
8 emma : type.
9 charles : type.
10 emmaCharlesMarried : type.
11
12 % Narrative actions
13 emmaSpendsYearsInConvent : type = emma * convent -o {@emma * !grace * !education
    }.
14 emmaReadsNovel : type = emma * novel -o {@emma * @escapism}.
15 emmaGoesToBall : type = emma * ball -o {@emma * @escapism}.
16 emmaMarries : type = emma * escapism * charles * grace -o {@emma * @charles
    * @emmaCharlesMarried}.
17
18
19 % Initial environment (resources + actions)
20 init : type = { @emma * convent * !novel * @charles * @ball
21     * @emmaSpendsYearsInConvent * emmaReadsNovel
22     * @emmaMarries * @emmaGoesToBall
23     }.
24
25 % Celf query
26 #query * * * 3 (init -o {emmaCharlesMarried}).

```

Fig. 6. Simple example of use of *TeLLer* provided by its creators

TeLLer had stopped at this point because their solution was about generating a syntax for exploring procedurality in narrative with a cleaner and shorter manner than with the already existing behavior trees. The current project could have ended up with this anti-innovative solution to solve the same trouble raised by another study in a more complex way. However, *ScripTale*, wanted to justify the use of trees in the States Editor with a meaningful output in the development of the story.

Narrative would be linked with dialogue in my implementation. How to make some structured field such as the dialogue up to an artificial intelligence was the obstacle that has been the hardest part of the current study until the last implementation of

ScripTale, and still is despite having found a solution. Classifying all the messages into different types of emotions could allow the tool to decide which message to pick up without any knowledge of the content but the emotion the message could cause to a character. To endow this feature with realism, not all the characters would be affected equally by a message, the effect had to be modified depending on the relationship between characters.

Realism is the key that is still found a limit for the tool. To achieve emotional realistic narrative behavior, the logic of the conversation was not contemplated. So characters must still have default sentences as variate as the designer wants to set them. However, they must be generic as conversations may not happen to make sense. This could be a further point of research for the current project. It was not implemented because a supposed to be simple tool would have become so complicated that AI would have less control over the decision taking process of the messages sent by the characters.

To conclude with this project, an observation has to be done. In an uncertain field such as Artificial Intelligence, the accomplishment of goals becomes dubious. They were technically achieved, however, better implementations can be done and studies still have a long way to improve in Procedural Narrative to get better results and really impressive conversational behavior totally up to the characters or agents of the AI.

Bibliography

- State of the art

AI meaning gameplay benefits [Online]. (2017, 15th September). Article, URL <<https://www.forbes.com/sites/quora/2017/09/15/how-will-artificial-intelligence-change-video-games/#6107d8dd2bb5>> [Consult 14th February 2018].

NPCs algorithms [Online]. (2017, 28th August). Blog, URL<<http://sitn.hms.harvard.edu/flash/2017/ai-video-games-toward-intelligent-games/>> [Consult 14th February 2018].

AI's effect on video games [Online]. (2018, 10th January). Blog, URL<<http://www.techradar.com/news/game-on-how-ai-is-transforming-video-games-forever>> [Consult 14th February 2018].

Future of AI in the video games field [Online]. (2017, 27th July). Blog, URL<<https://www.theatlantic.com/technology/archive/2017/07/musk-vs-zuck/535077>> [Consult 15th February 2018].

Visual improvements [Online]. (2018, 10th January). Blog, URL<<https://www.techradar.com/news/tim-sweeney-refreshing-consoles-every-several-years-keeps-everyone-happy/2#article-body>> [Consult 17th February 2018].

Voice recognition video game [Online]. (2017, 2nd February). Article, URL<<https://www.bbc.co.uk/taster/pilots/inspection-chamber>> [Consult 17th February 2018].

Destiny 2 Alexa [Online]. (2017, 30th November). Article, URL<<https://www.polygon.com/2017/11/30/16719532/destiny-2-amazon-alexa-voice-commands-ghost-shell>> [Consult 17th February 2018].

- Procedural content generation:

Green Dale. (2016). *Procedural Content Generation for C++ Game Development*.

UK: Packt Publishing. Community Experience Distilled.

Watkins Ryan. (2016). *Procedural Content Generation for Unity Game Development*.

UK: Packt Publishing. Community Experience Distilled.

Areas in which PCG can be applied [Online]. (2017, 14th July). Video, URL

<https://www.youtube.com/watch?time_continue=2&v=0vST4lvG5CE> [Consult 8th February 2018].

PCG implementation [Online]. (GDC Europe 2014). Conference content, URL

<<https://www.gdcvault.com/play/1020868/Games-with-Freedom-Programming-Procedural>> [Consult 8th February 2018].

Angelina [Online]. (2017, 29th November). Blog, URL

<<https://www.technologyreview.com/s/609482/ai-is-dreaming-up-new-kinds-of-video-games/>> [Consult 9th February 2018].

Games made by Angelina [Online]. (2018, 15th January). Web page, URL

<<http://www.gamesbyangelina.org/>> [Consult 9th February 2018].

- *Procedural narrative information:*

Narrative Structure [Online]. (2017, 6th September). Blog, URL

<http://www.gamasutra.com/blogs/AndrzejMarczewski/20170609/299705/Narrative_Atoms_and_the_Soap_Heros_Journey.php> [Consult 8th February 2018].

Practices for Procedural Narrative Generation [Online]. (2017, 12th May). Video, URL

<<https://www.youtube.com/watch?v=k2rgzZ2WXKo>> [Consult 7th February 2018].

Practices for Procedural Narrative Generation [Online]. (2016, 10th January). Video,

URL <<https://www.youtube.com/watch?v=p40p0AVUH70>> [Consult 7th February 2018].

- Development

Schudlo, Nicholas A.. (2014). *Development of an Emergent Narrative Generation Architecture for Videogames*.

Bosser A., Ferreira J. F., and Cavazza M., and Martens C.. *Linear Logic Programming for Narrative Generation* [Online]. Research, URL
<<https://www.cs.cmu.edu/~cmartens/lpnmr13.pdf>> [Consult 21st March 2018].

Goal Oriented Action Planning for a Smarter AI [Online]. (2014, 23rd April). Tutorial, URL
<<https://gamedevelopment.tutsplus.com/tutorials/goal-oriented-action-planning-for-a-smarter-ai--cms-20793>> [Consult 20th April 2018].

Annexes

- **Annex 1: ScripTale.unitypackage**

Annex 1, named ScripTale.unitypackage can be found in the same folder as the current memory. It consists on the tool itself, the resultant package to be imported to unity so as to make use of the tool.

- **Annex 2: Readme.md**

Annex 2, named Readme.md can be found in the same folder as the current memory. It consists on the README of the tool, the instructions of use.